

## Real-Time Control Systems for Robots

M. L. Fitzgerald  
Anthony J. Barbera  
J. S. Albus  
National Bureau of Standards  
Gaithersburg, Maryland

### ABSTRACT

The advent of low cost microcomputer hardware has provided the capability for sophisticated control of robot machinery, allowing such features as straight line trajectory motions, sensory interaction, workstation integration, external database access and off-line programming.

The foundation of these capabilities lies in the architecture of the computer hardware and software system. This real-time control architecture must provide the framework for a logical partitioning and structuring of the control tasks into functionally separate modules that are bounded by well-defined data interfaces. It is through this architecture definition that robot controllers can be designed and built that allow enhancement of control strategies through upgrades in isolated modules, addition of sensors for real-time modification of behavior, and integration into workstations communicating with external knowledge bases through the use of defined data interfaces.

This paper will report on such an architecture for real-time control. The fundamental building block is the input-process-output structure. From this is derived a generic control level that can be stacked into multiple levels to provide an environment for hierarchical decomposition of the task. The more complex the task, the more levels that are required. Efficiency and reliability are obtained through a highly interactive user interface with diagnostic probing and display capability that allows quick evaluation and modification as required to accomplish the intended tasks.

---

Delivered at 1985 SPI National  
Plastics Exposition Conference  
McCormick Place, Chicago

June 19th  
Robotics

## I. INTRODUCTION

The ability of a robot to carry out a programmed task successfully is dependent, to a large part, on the control system that runs it. Since robots were first introduced, 15 or 20 years ago, their control systems have improved dramatically, primarily due to the introduction of high-speed, low-cost, very powerful computer systems and the corresponding sophisticated software required to generate real-time control. This paper will attempt to address the development of these control systems and partition them into four major classes. Each of these will be described by a set of characteristics, with some perceived advantages and disadvantages that help distinguish one from the other. These classes are arbitrary in their partitioning but help to demonstrate the range of capabilities and the significant contributing factors that allow certain capabilities.

It will be shown in the four different classes of controllers outlined in this paper, that the capabilities of the robots increase with the use of sensory information and with their integration into larger systems. However, these additional capabilities come at the cost of more complexity in the information processing. This paper will also describe the research into robot controller architectures presently being done at the National Bureau of Standards.

The major requirements that these systems will be measured against are modularity, flexibility and reliability. Modularity describes the ability to easily add new capabilities or components to the system such as different sensors, different robots with different mechanical configurations, and the ability to integrate the controller into workstations of which the robot is only one part. That is, modularity is the ability to treat various functional parts of a large system as distinct components which can be replaced with other modules as long as they carry out the function and meet the interfaces.

Flexibility describes the control system's ability to be reprogrammed to handle different parts in the environment; to do different tasks; to handle different work environments in the sense of different trajectory

paths, different avoidance maneuvers, different approach and departure paths to and from the various work pieces. Thus, flexibility is the ability to deal with changes in the application task of the robot.

Reliability is concerned with the basic ability of the user to understand the system that he has created, so that he can ensure that it will do what is intended. To this end it is important to have a system comprehensible to the user at all levels of detail, to have extensive diagnostic capabilities, and to be interactive in the sense that the user can probe the system, can put in certain conditions, and quickly see what the response of the system will be.

## II. CLASSIFICATIONS OF ROBOT CONTROLLERS

The four major classifications of robot controllers that will be made are: 1) the record/play-back, 2) computer assisted record/play-back, 3) programming language based control system, and 4) fourth generation controllers.

### II.1. Record/Playback Controller

The first type of controller, record/playback, is basically a system that looks like a tape recorder. The user can lead the robot through a series of points in space, recording each of these points, essentially on a tape recorder, to be played back at some later time. An advantage of this system is that the user interactively programs the robot and sees exactly what he is telling it to do. There is a single sequence through space that the robot is allowed to follow that is visible to the user during programming and it is this sequence, and only this sequence, that is repeated over and over again.

A disadvantage of this type of system is that it can be very tedious for the user to program precision tasks with the robot. Most robots include revolute joints. In this joint space it is not easy to control straight-line motion of the robot, or to cause the tool-tip of the robot to move along some arbitrary path by moving these rotating joints. Thus, very precise positioning of the end point and the orientation of the robot can be very tedious, time consuming and difficult.

Another disadvantage of the system lies in the inability to easily modify what the robot has been programmed to do. The points are played back in the same sequence that they were recorded. This means that if the user wants the robot to move to a slightly different intermediate point, he must again lead the robot through the sequence up to that point, re-record the new position, and then the robot can play back the new sequence. It is not possible to insert additional points, nor delete points out of the sequence, so there is a lack of flexibility in being able to program new tasks or to handle different parts with the system. Moreover, a stored program taught on one robot cannot be transferred to another robot of the same type and have the same task trajectories executed. This is mainly due to the differences in link lengths, joint orientations, slightly different placements of joint position sensors, and different values of gain parameters in the servo systems.

In addition, this particular type of controller has no decision capability. It is not capable of branching to carry out one of two alternative task sequences; it must follow a single sequence of recorded points. This means that it is not possible for the controller to interact with simple sensors or other devices such as workstation controllers or machine-tool controllers. Thus, the tasks that this type of control system can deal with are only those tasks that can be accurately and reliably defined by a sequence of fixed points that the robot must follow through space. If the error that the robot has in its repeatability in getting to these points, added to the misalignment of the parts, results in an offset greater than that required for successful completion of the task, then this type of controller cannot be used.

## II.2. Computer Assisted Record/Playback

The second type controller is that of computer assisted record/play-back. A computer that performs certain limited basic functions has been added to enhance the user's ability to program the robot. The main feature provided by computer-assist is the ability to calculate the transformation from some defined coordinate system to the joint coordinates of the robot. These controllers allow the user to program straight line motions, motions about a tool-point axis and other trajectories which reference some relatively robot

independent coordinate system. That is, the user can describe the position or motion of the tool point in a convenient representation, and that position will be transformed into the proper joint values for the robot. The user still has the advantage of being able to teach-program the robot, thus interactively seeing what the robot is going to do during the execution of the task. But now, programming is easier since the user, through the joy-stick, commands motions along straight-line paths, and orients the end effector for precise positioning more easily than by joint space manipulations.

These controllers provide a higher degree of trajectory control during execution than the record/playback controllers. That is, even though the programming still consists of recording only the end points of the trajectories, the executed trajectories can be straight lines rather than the arc type motions about the rotary joints. This straight line motion is due to the real-time computation of the coordinate transformations. The end points are stored in the form of some workspace coordinates such as cartesian, cylindrical or spherical. Intermediate trajectory points are calculated during execution and transformed to the corresponding robot joint coordinate values. This results in controlled-trajectory motion in the workspace coordinate system.

A further feature provided by computer assistance is the ability to edit the programs that have been generated. The user is allowed to modify the trajectory paths of the robot by the addition or deletion of two or more points without being forced to re-record the remaining points of the sequence.

In addition, computer assisted controllers usually provide the ability to do simple branching between different sequences of paths based upon some input value. In this manner the controller for the robot can use simple binary switches to determine alternative actions. For instance, a sensor may be used to distinguish between two differently shaped parts. The robot is programmed with two different procedures, one for each of the parts. Depending upon the binary value read in from the sensor, the controller branches to one or the other of these sequences.

Interlock capabilities are also provided. The controller may be halted at a defined point in the program until a signal from an external device indicates the appropriate time to continue. For example, when a machine tool has finished a cut, it may indicate through a switch that the controller can now send the robot into the work environment to pick up the part. A computer assisted controller may also generate output signals that can be used for a positive handshake interlock with other devices.

The interaction with sensors is, however, very limited with this type of controller. It can handle the binary type of sensors, but does not have sufficient programming capability available to the user to allow for the branching and analysis required by the sophisticated data that would come back from complex sensors such as vision, force and torque, or proximity.

### II.3. Programming Language Based Control System

The third type of controller is supported by a full computer programming language. These are the computer based programming systems that have become available in the past few years (2-4). They offer the user the ability to do complex decision programming and processing. Control programs can be written so that data from different sensors such as vision, proximity, force and torque, can be processed and decisions made based on this information. This programming capability includes the ability to edit the programs and do essentially total off-line programming if the coordinates of the workspace are well known. In most cases they are not precisely known, therefore some method of teach-programming must still reside with these systems to move the robot to certain points in space for a particular task, record the coordinates of these points in joint space and transform them back into cartesian space. Once this data is entered into the system, however, it can be edited, moved, or modified. Approach paths to a point can be defined by algorithms that will determine how to vary the path of the robot based on available sensory data. Trajectory paths between points may be constrained to follow a predefined surface, rather than merely straight line motions. The task can essentially be constructed as a computer program.

A number of convenient library routines or primitives may be provided with the system. Usually there exists a trajectory algorithm used to generate paths in space. The ability to do general coordinate transformation matrix operations may also be available. Using these matrix operators, different end-effector configurations and multiple coordinate frames can be specified in the system to calculate the position of the end-effector and to control the robot based on the motion described relative to different objects such as trays, machine tools, etc. Delta motions - e.g. five inches in the X direction - provide convenient ways of specifying path modifications to occur in real-time based on incoming sensory data.

Because of the high level programming language provided, these type of systems have come to be used as the highest level controller in the environment, taking on the aspect of the coordinator and controller for a whole workstation. These robot controllers have full computer capability which many of the other components or devices in the manufacturing environment do not. Thus, when a robot controller of this type is to be used with machine tools, fixturing, and transport devices, more than likely the supervisory and coordinating programs will be written inside the robot controller because of the programming language capability to handle these interactions and to define what the overall task should be.

The disadvantage of this system is one that is shared by all large programming systems. That is, without a very rigorous high-level architecture to which this system is confined in its growth and complexity, it can quickly become a large set of software programs that are hard to understand in their interactions, difficult to trace in their effects from modifications, unpredictable in their execution behavior, and, in general unmanagable, and therefore unreliable as a system. Most present robot controllers were not intended to be part of a larger system. This type of controller provides the user with tools for programming the robot. However, as the tasks get more and more complicated, there is no guideline, there is no higher level structure, to control the growth of these programs in any way that will allow them to be extensible and to have interface capability to other systems. It is also difficult to control the growth in a way that the user can feel confident in the reliability of the system.

The tendency to make this type of system the highest level controller also tends to create a system that may not be easily integrated with yet larger systems. The robot, which is a mechanical device of about the same task importance in the overall system as a machine tool, now has in its controller the control of the robot and the the supervisory control to coordinate the robot with the machine tool and other devices in the workstation. Interactions with the rest of an automated factory must now occur through the robot controller in order to coordinate the command request into the workstation for the different parts to be made. This includes the movement of new parts and new tools in and out of the workstation, accessing the data base for the parts program for the NC tool as well as for parts descriptions from CAD data bases for use with sensory processing algorithms. Since the workstation coordinating and supervisory programs reside within the robot controller, the interactions of the other workstation components may not be clearly delineated from the robot's operations, which will result in a system that is difficult to understand or reprogram.

#### II.4. Fourth Generation Controllers

Fourth generation controllers are an area of research at a number of institutions today. The philosophy behind these controllers is that the robot controller is but a modular component of a larger integrated system. As such, it will have certain constraints imposed upon the design and construction of the processing algorithms that occur within it - the type of interfaces, the information paths, the data flow requirements - so that it acts as an integral but modular component of this larger automated system. This type of controller provides all of the same capabilities that are available with the three previously described controllers, including the advantages that have been obtained by adding computers to the system. It also provides the further advantage of creating a framework for a flexible system, where different parts and workpieces can be handled in different work environments. These controllers are based on system architectures that emphasize system modularity where the functions to be performed are partitioned and defined in a highly structured manner, creating well-bounded modules with defined data interfaces that will allow for different sensors, different robots, and even the controller itself, to be plugged into different configurations.

A well structured system imbeds information about the operation of the system by the very organization of its components. Information maintained in a rigid structure helps increase the reliability of the system since it makes the system more comprehensible to the user with everything kept in a specified place. The structure provides a mechanism whereby the user, much like using a road map or a file cabinet, knows where to find a certain processing task in order to make additions to that task, modifications to it, or to do diagnostic analysis on it. Generic modules greatly reduce the complexity of the system by requiring the user to understand only a small number of processing modules which are repeated through the entire system.

## II.5 Summary

Figure II.1 summarizes the advantages and disadvantages of the four types of controllers.

## III. NBS CONTROL SYSTEM ARCHITECTURE

The National Bureau of Standards has concentrated various research efforts in the area of integrated control structures for fourth generation controllers. Its interests are in defining the framework - the organizational structure and processing requirements - to design an integrated, totally automated factory system (5-9). A robot controller is but one component, one module, in this integrated system. It has to interface into the overall system as a component that will share its data with the system. It relies on an external data base administrator to provide the information generated by other components in the system. The basic work to date has been in defining the system architecture. The architecture (10,11) has two main components - processing and structural. The processing architecture defines the component modules of the system, how the internal functioning of each module is accomplished and how and when information is exchanged between modules. The structural architecture describes how each of the modules interrelates in order to have the system exhibit the desired real-time sensory-interactive behaviour.

### III.1 The Processing Architecture

It has been found through experimentation at NBS over a number of years, that certain basic techniques have greatly aided the development of these complex control systems. The first is the use of the input-process-output structure to create well-bounded functional modules with specified data interfaces. The second is to use generic structures wherever possible within the components of the system. The third technique is the use of a common memory based communication mechanism to move data among all the system components, and the last is to execute all components in the system on a repetitive cycle.

#### III.1.1 Input-Process-Output

The input-process-output structure (Figure III.1) is the fundamental building block upon which all the processing within the system will be based. The goal is to provide a system of functionally well defined modules with interfaces between them that provides all the capabilities mentioned previously. The basic model becomes that of a functional module that processes its input data set to generate a resulting set of output data. The function that each module performs should be independent of other processing that might occur within the system, with information required between modules being passed only through the interface data sets. New capabilities may be added to the system - new sensors, new algorithms for directing robot trajectories, new end-effectors - all by providing modules to handle these components, and integrating them with the other components of the system through their data interfaces. The robot controller itself is a component defined in this manner and may be integrated with a higher level controller, such as a workstation, by meeting the interface data requirements that are defined for it.

A great deal of care has to be given to the design of the overall system based on these type of modules. It is not difficult to generate a system of modules that are so completely interwoven in their function that the modularity is lost and that it is impossible to clearly define their interfaces. These types of systems are to be avoided since a user cannot understand the function of one of the modules without having to understand what all of its interrelated modules do as well. Thus, it is extremely important to emphasize the need to define well

bounded functional modules which are as cleanly separated as possible from all other modules in the system, so that the function of the module can be understood by simply looking at its input data and output data and a description of its processing. In addition, this technique facilitates module testing and debugging as, by supplying input data sets to a module, it may be tested in isolation, either before being added to the system, or afterwards, to diagnose a problem.

### III.1.2 Generic Processing Structures

Having structured all the components of the system into the input-process-output format, a problem develops in that the number of functional modules themselves becomes large making it difficult for the user to interact with the system. A primary method of dealing with this problem is to develop generic processing structures. That is, attempt to make the operation of the modules look the same, so that they appear to be replication of one processing format. Even though there are a large number of components, the user is not overwhelmed because, having understood one of the components, he knows that he can understand the operation of the remainder of the components. This use of generic structures is extremely important in the development of a successful system.

The use of the input-process-output structure is a type of generic processing. However, here it is desired to specify a generic processing within the modules. This structure can be viewed as three separate processing operations: preprocessing, decision processing, and post-processing (Figure III.2).

The decision processing is the major decision mechanism used to identify the appropriate procedures that will execute based on a few high-level variables. It specifies the various test conditions and the corresponding output procedures to be executed if those test conditions are satisfied. Preprocessing evaluates, scales, reduces, and transforms the input data into the more appropriate set of variables used by the decision processing. Postprocessing picks up the extraneous, but necessary, additional processing required. It performs such functions as saving internal variables or reformatting algorithm results for output or transfer to other modules.

This technique of using a generic processing format does much to ease the user's interaction and comprehension of each unit module and can be applied to larger system components as well.

### III.1.3 Common Memory Based Communications

The use of the above generic structures creates a system of separate components that interact only through their input and output data interfaces. To connect any two components, the output data of one must become the input data of the other. This implies that there must be some method to transfer data between these components. A communications mechanism has been defined that uses a common memory area. A copy of the output data is moved to a duplicate buffer area in common memory and from there moved into the input buffer of the other module (Figure III.3).

This type of communications provides the system with a number of advantages. First, the addition or deletion of processes or components into the system is simplified. Data is transferred between components through an independent link, therefore each component may be developed in isolation by supplying appropriate test values to its input data set. Once the modules have been tested, they may be integrated by allowing the communications mechanism to supply their data. It means that even when the system is integrated, each of the processing modules executes as an isolated process bounded by its data interfaces. Whether the input data to a module is supplied by process B or new process B' is immaterial. Second, the buffering of data in common memory is a convenient structure because it eliminates any need for synchronization between sending and receiving modules. Data is moved from the first module to common memory whenever that module is ready to send it, and moves from common memory to the second module whenever that module is ready to receive it, totally asynchronous with the first module's execution. This helps simplify the implementation of these processes on multiple distributed computers.

Moreover, the existence of the duplicate copy of these system buffers in common memory makes available to an additional process, such as a diagnostic process, the present values of all of the critical variables in the system. If the communications process is executed periodically, then there is an interval when the data in

the common memory buffers are not being altered. During this time, a diagnostic process can capture and display this data for user evaluation of the real-time internal behavior of the system.

#### III.1.4 Repetitive Cycling Execution

The real-time aspect of control is based on the concept of producing a response to changes in input data soon enough for that response to be effective. If a control cycle ( in which the input data is sampled and the output response generated ) is repeated at a sufficiently fast rate, the system will provide apparent continuous control in real-time.

The communication mechanism described above can be used to define this repetitive processing through the use of a periodic synchronization pulse. The communication process can move all the available data to all the processes that are ready to process. After the data transfer is done, those processes will execute acting on the data present in their input buffers. The communications process itself runs periodically off a real-time clock. This defines the basic control cycle. Every cycle, after communications has transferred the data, each process that is ready sample its data and generate its output response (Figure III.4). All of the system processing occurs each cycle based on the current input data set. This means that the system is not event driven in the sense that an event generates an interrupt which modifies the control flow. All of the relevant variables are sampled each cycle. That information is processed through totally deterministic preprogrammed algorithms. The system therefore offers a very deterministic, well-defined view of a control algorithm that executes in real-time and further, can be executed offline for diagnostic testing by supplying the data to its interface.

#### III.1.5 Benefits of the Processing Architecture

This processing architecture provides a framework for breaking the processing of the system into component modules, and to control the design of those modules. The technique of building the system as well-bounded functions with clearly defined data interfaces directs the total modularity of the system. Different components and capabilities can be added to the system by adding new modules that perform the desired function

through a well defined data interface.

By using generic processing structures in the system, the reliability of that system is increased. The user will be able to understand how to build the system to make it do what he wants it to do and to understand how it is working. Moreover, he will be able to modify or add pieces to the system in a structured manner that will ensure that those changes will still maintain the correct operation of the system.

The common memory communication mechanism along with the well-bounded modules with data interfaces helps provide a system with a highly interactive user-friendly programming environment. A typical problem with large complex systems is that even though the individual subcomponents may be understood and that their action well defined and well specified, their behavior when assembled together in large systems is often unexpected. By using a diagnostic tool to display the system parameters from common memory during execution, the user can more easily understand, and therefore improve, on the system performance.

The repetitive sample control structure allows for a totally deterministic control system in which components can be evaluated piece by piece and which will behave offline, in the non-real-time non-integrated condition, as they will in the entire integrated system.

### III.2 The Structural Architecture

A structural architecture for a fourth generation robot controller that is a component part of a larger system architecture for an entire automated factory has been developed at NBS. The robot controller is one module between a workstation controller and a robot joint controller, with access to sensory processing and world model data relevant to the type of task that it will handle (Figure III.5). Several implementation techniques have been developed which, when applied with the methods of the processing architecture, organize complex information processing into a flexible system which will effectively provide sensory-interactive real-time control (12). These include the use of generic processing levels that perform hierarchical task decomposition, the specification of interfaces between control levels, and the separation of task and data for all components of the system.

### III.2.1 Generic Task Decomposition Levels

The overall structural architecture is based on a set of generic levels, generic processing modules, that perform the hierarchical task decomposition of the input commands for the robot controller. Figure III.6 shows a robot controller defined as a set of three processing levels. The concurrent operation of these control levels provides a stepwise decomposition of high level tasks into successively simpler and simpler component subtasks. By only requiring each level to decompose the task a little further in the next lower set of subtasks, it is relatively simple to comprehend and manage the control function of each level.

It is very desirable to make these control levels as generic as possible - that is have essentially the identical processing structures - so that having understood how one of the control levels performs, it is fairly straightforward to understand how all the others perform. This allows the user, as well as the system designer, to create systems that are reliable, modular, and to develop plug compatible interfaces through the data sets that bound each of these modules. The generic control level is defined as a process that will do a partial task decomposition of its input command into a sequence of not more than seven or eight subcommands. The level is thus structured in the input-process-output format.

Keeping with the processing architecture previously discussed, processing within the level has been further defined and partitioned into a preprocessing, a decision-processing, and a post-processing phase. The major decision processing that occurs within a level is to take the input command, relevant data from the sensory processing/world model, and status from the level below, and decide on the next output command, output status and request for the sensory system (Figure III.6). This is the major decision processing that occurs within this control level. The preprocessing for a control level is used to take the input data and convert or scale it to create variables that will simplify the determination of the next subcommand. The post-processing phase is used to perform utility functions such as reformatting data and maintaining internal status.

Each of the different control levels can be described by this generic type of processing. The functional processing within each level identifies for the user where appropriate information should be contained. This increases the ease with which the user can upgrade the capabilities of the system, such as deciding where to add additional sensors. The generic structure of each of these major components ( the levels ) provides the consistent framework for the user to know where each module belongs within the system.

For example, the robot controller in Figure III.6 shows a three level task decomposition. The TASK level takes input commands from the workstation and generates subcommands that define major end points for the robot motions. The ELEMENTAL-MOVE (E-MOVE) level generates trajectory segment goal points that define a complete path to an end point. The PRIMITIVE level generates each of the intermediate path points along the trajectory segments required to move the robot to the goal points.

Consider the addition of a vision system that is capable of providing ranging, feature detection and object recognition. Identification of where these different types of information should be added to the system is made straightforward through the system structure. Ranging information is appropriate at the PRIMITIVE level since it is doing high speed servoing of the robot motion. Full object recognition is most appropriately added at the TASK level since it is concerned with major motions of the robot. The actual algorithms for handling this sensor data at each level are easily included into the code because of the well-defined framework of the processing structure.

### III.2.2 Control Interfaces

Each generic control level is connected through data interfaces to three other components within the system. There is an interface to a control level above that performs a higher level decomposition. This interface is composed of two buffers - a command buffer from and a status buffer to the higher level. In like manner, there is a similar interface to a control level below that consists of a command buffer down and status buffer back from this lower level. There is an interface horizontally to the sensory processing/world model system, which includes the request buffer to this system and the feedback response buffer.

These three major interfaces surround each of the generic control levels. Viewed differently, they act as as a set of input data that is processed to generate a set of output data. The input data is the command in from the level above, the sensory processing/world model feedback, and the status coming in from the level below. The function of the control level is to take those three major data interfaces and generate the output data which consists of a command out, a sensory processing/world model request out, and a status out to the level above.

The interfaces that exist between these control levels then, become the interfaces that allow for the type of modularity required in the system (Figure III.5). The generic task controller decomposes the task into simpler subtasks until eventually the task is described in terms of the sequence of positions and orientations of the end-effector in space. This information can then be passed across the interface to a robot joint controller. This interface can potentially be in a robot independent form. The function of the robot joint controller then becomes the transformation of the interface information into the joint values for a particular robot that are required to orient and position the end-effector at the commanded pose.

The information that comes from the workstation controller often defines an interface of the same type. That is, the commands that are issued from the workstation controller are generic to the task, not the particular robot or robot controller that's involved. All that is required is that those input commands can be be decomposed by the particular robot controller and status reported back to the workstation according to the defined interface information.

The interface to the sensory processing/world model is really a series of interfaces, one to each of the different control levels found in the architecture. The high level information, like that of whole object recognition is an interface into the TASK level. An interface containing feature or force and torque information would be to the E-MOVE level since this type of feedback usually affects the real-time modification of trajectory segments. An interface describing perhaps touch or proximity or some other component of force and torque might be data that is used by the PRIMITIVE level to modify intermediate points of the trajectory, servoing the robot's motion every control cycle.

These interfaces offer the potential for providing plug compatibility with many different types of sensors or other components with the controller. If the controller consists of functional modules bounded by clearly defined interfaces, and if a proper communication structure is set up such that these are not directly coupled with the elements that are developing the information on the other side of the interfaces, then these blocks can become different components that can be plugged together in different ways without resulting in a bundled system.

### III.2.3 Task and Data Independence

In order for the system to display the type of flexibility required in an automated factory, it is important that the data, as much as possible, be totally separated from the actual algorithms for each of the modules. All of the processing within the system should be done in terms of symbolic variables whose values can be supplied at execution time. There should be no constants imbedded within the code.

For example, the transfer task - to move an object from one point to another - is defined by the decomposition algorithm that has the following subcomponent actions: move to some source location, pick up an object, move to some destination location, set down the object, and withdraw. That task decomposition is independent of whatever the object is, and wherever that object is to be picked up and put down. The particular numeric representation that defines the object and the locations in space it is to be moved through, is the data that must be separated entirely from the task. Using this technique, it is possible to think of generic controllers that are programmed to carry out whole classes of task decompositions which accept at execution time the data that specifies the particular objects, particular locations, particular workstations, particular robots, that have to be used for one particular task.

Thus, the data in the system that defines the specific instances of the parts to be handled, the trajectory paths through the workspace etc, is totally separated from the algorithms. It can be created off-line, perhaps developed through a CAD system, or through some other off-line programming environment, to be loaded into the system when the particular task is to be

executed. The partitioning of the control algorithms from this data will provide a mechanism for the flexibility that is desired in the final system. That is, the algorithms for a particular task can be transferred to different workstations, and loaded with the data pertinent to that workstation and that task can still be accomplished even though it had not been "taught" or "programmed", in the traditional sense, for this specific part on the new workstation.

#### III.2.4 Summary

The original requirements that were placed on the system were modularity, flexibility, and reliability. The system is modular if different sensors could be added for a particular task, if the controller can interface to different robots allowing one robot to be substituted for another in a workstation, if one trajectory module could be substituted for another, or if different workstation controllers could talk to the same robot controller. This type of modularity, being able to plug these units together in much the same way that a stereo system can be configured with different components, requires a very strict and rigid architecture and structure of the system that allows components to be developed as independent functional modules with well defined data interfaces. The system must also be structured so that where each component is to be added is readily apparent. In addition, the system must provide a mechanism to execute each of these individual components as a unit, transferring data among them and still maintaining the timing requirements to provide stable real-time behaviour.

Flexibility is the ability to handle the multiplicity of work pieces, to easily program different tasks, and to define different trajectory paths through space involved with different tasks. This ability requires a structure where only the data that describes the numerical values of the new coordinates of the system, or sizes of the parts, or locations of trajectory points need be altered. Ideally, these can be altered from some higher level off-line system, such as an overall CAD system. The separation of task algorithms from the data creates a system where it is possible to think of off-line programming through data structures without having to reprogram controllers to handle the task to be accomplished. This adds a tremendous amount of flexibility in the system, although it creates

additional complexity in trying to develop these type of structures.

Reliability is provided by an architecture that so identifies the components of the system that the user can quickly find and locate and zoom in on any aspect of the system to understand exactly how its working at some level of detail. The user must interactively be able to identify how each function reacts to changes in input data, be able to set up conditions so that the system can run in real-time, with all of the processes executing in parallel and with the user still able to track and display and observe the different internal states of the system.

Much of the processing and structural architecture described has been based on the concept that people can only deal with a small amount of information at one time. The system is built as a set of modules, each of which totally bounds a certain small part of the function and being able to understand the workings of all of the parts, enhances the users ability to interact with, deal with and understand the function of the integrated system. In addition, by maintaining all of the modules in a generic structure, a generic framework, to process information in the same way, to use the same major data inputs and outputs, creates a very familiar environment at each level to the user. This will ultimately result in a more reliable system.

This whole area of fourth generation controllers is still in need of additional research before its feasibility in totally automated systems of reasonable functional complexity can be verified.

#### Acknowledgements

This work is partially supported by funding from the Navy Manufacturing Technology Program.

This article was prepared by United States Government employees as part of their official duties and is therefore a work of the U.S. Government and not subject to copyright.

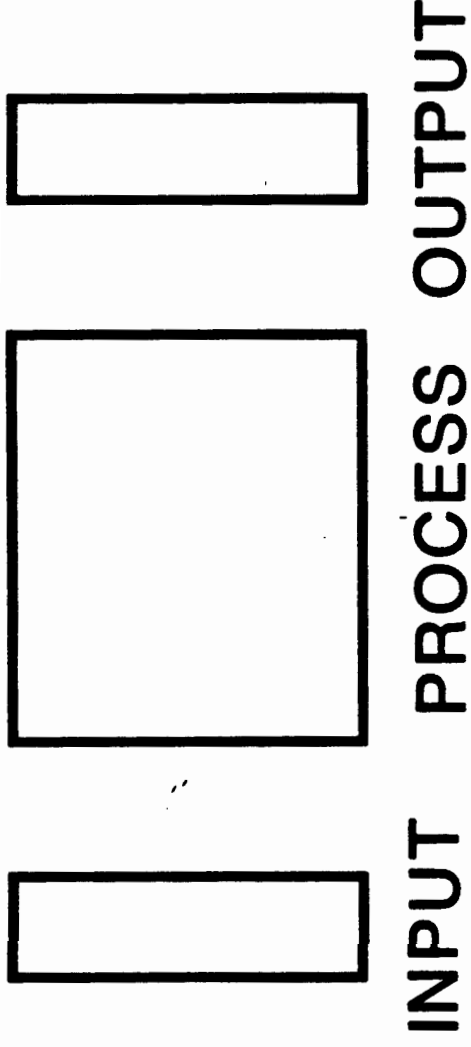
## References

1. Bonner, S. , Shin, K.G., "A Comparative Study of Robot Languages", Computer, Vol. 15, No. 12, December 1982.
2. Shimano, B.E., Geschke, C.C., Spaulding, C.H., "VAL-II: A New Robot Control System for Automatic Manufacturing", Proceedings of IEEE International Conference on Robotics, Atlanta, GA March 1984.
3. VanderBrug, G.J., "RAIL: A Language for Vision and Robotics", Proceedings IEEE Computer Society COMPSAC, New York, 1981.
4. Taylor, R.J., Summers, P.D., Meyer, J.M., "AML: A Manufacturing Language", International Journal of Robotics Research, Vol. 1, No. 3, Fall 1982.
5. Albus, J.S., Barbera, A.J., Nagel, R.N., "Theory and Practice of Hierarchical Control", 23rd IEEE Computer Society International Conference, Sept. 1981.
6. Simpson, J.A., Hocken, R.J., and Albus, J.S., "The Automated Manufacturing Research Facility of the National Bureau of Standards", Journal of Manufacturing Systems, Vol. 1(1):17-32, 1982.
7. Albus, J.S., McLean, C.R., Barbera, A.J., Fitzgerald, M.L, "Hierarchical Control for Robots in an Automated Factory", 13th ISIR/Robots 7 Symposium on System Theory, Chicago, IL, April 1983.
8. McLean, C.R., Mitchell, M.J., Barkmeyer, E., "A Computing Architecture for Small Batch Manufacturing", IEEE Spectrum Magazine, 1983, p59-64.
9. Furlani, C.M., Kent, E.W., "The Automated Manufacturing Research Facility of the National Bureau of Standards", Summer Simulation Conference, Vancouver, B.C., July 11-13, 1983.
10. Barbera, A.J., Fitzgerald, M.L., Albus, J.S., Haynes, L.S., "A Language Independent Superstructure for Implementing Real-Time Control Systems", International Workshop on High-Level Computer Architecture, May 1984.

11. Barbera, A.J., Fitzgerald, M.L., Albus, J.S.,  
"Concepts for a Real-Time Sensory-Interactive Control  
System Architecture", Proceedings of the 14th  
Southeastern Symposium on System Theory, April 1982.

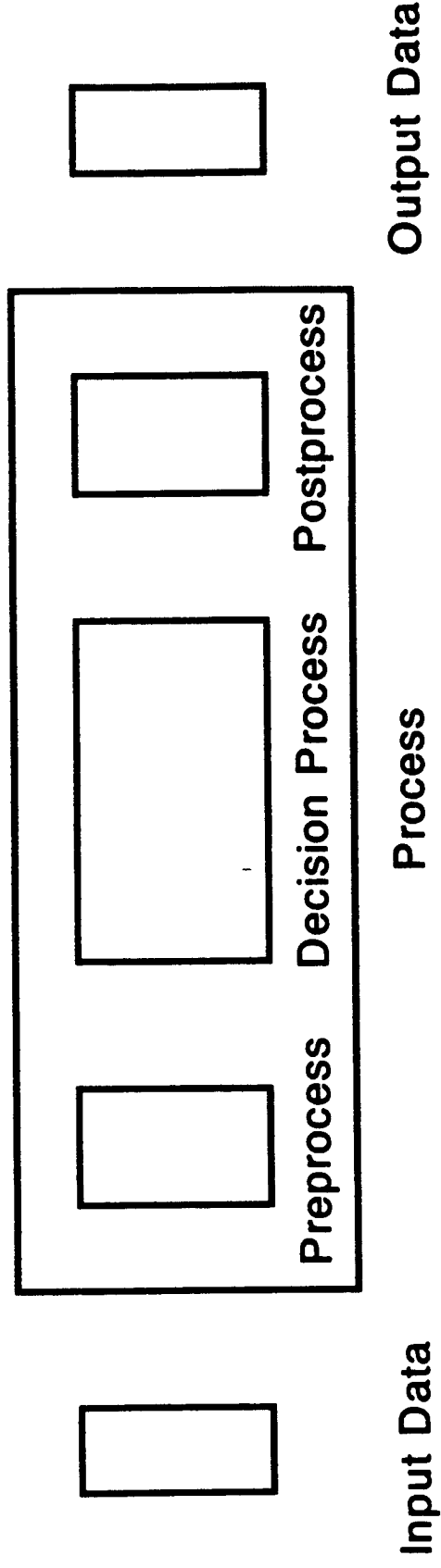
12. Barbera, A.J., Fitzgerald, M.L., Albus, J.S.,  
Haynes, J.S., "RCS: The NBS Real-Time Control System",  
Proceedings of Robots 8 Conference, Detroit, June 1984.

	<u>ADVANTAGES</u>	<u>DISADVANTAGES</u>
RECORD/PLAYBACK	<ul style="list-style-type: none"> <li>* Highly Interactive</li> <li>* Easy to see what program will do</li> </ul>	<ul style="list-style-type: none"> <li>* No control over path motion</li> <li>* Hard to alter programs</li> <li>* No sensory interaction</li> <li>* Program is task and robot specific, not portable</li> <li>* Must be programmed online</li> <li>* Minimal error detection</li> <li>* Precision programming tedious</li> </ul>
COMPUTER ASSISTED RECORD/PLAYBACK	<ul style="list-style-type: none"> <li>* Interactive</li> <li>* Binary sensor capability</li> <li>* More convenient user interface thru joystick</li> <li>* Programs may be edited offline</li> <li>* Path motion defined</li> </ul>	<ul style="list-style-type: none"> <li>* Only minimal sensor interaction</li> <li>* May be robot/task specific, not portable</li> <li>* Minimal error recovery</li> </ul>
PROGRAMMING LANGUAGE BASED CONTROLLER	<ul style="list-style-type: none"> <li>* Full programming language</li> <li>* Full range of sensor capabilities</li> <li>* Can be modified or reprogrammed</li> <li>* May be programmed offline</li> </ul>	<ul style="list-style-type: none"> <li>* More complex to program</li> <li>* Not designed to interface to high level system</li> <li>* Debugging difficult</li> <li>* Much effort required building data driven generic tasks and interfacing to external systems</li> </ul>
4TH GENERATION CONTROLLER	<ul style="list-style-type: none"> <li>* Interactive</li> <li>* Easier to add new capabilities</li> <li>* Transportable programs</li> <li>* Task oriented programming</li> <li>* Generic task programming/ Data driven</li> <li>* Full diagnostic capability</li> <li>* Capable of integration with larger systems</li> </ul>	<ul style="list-style-type: none"> <li>* Complex to program</li> <li>* Requires sophisticated computer hardware and software</li> <li>* Research area, not available</li> </ul>

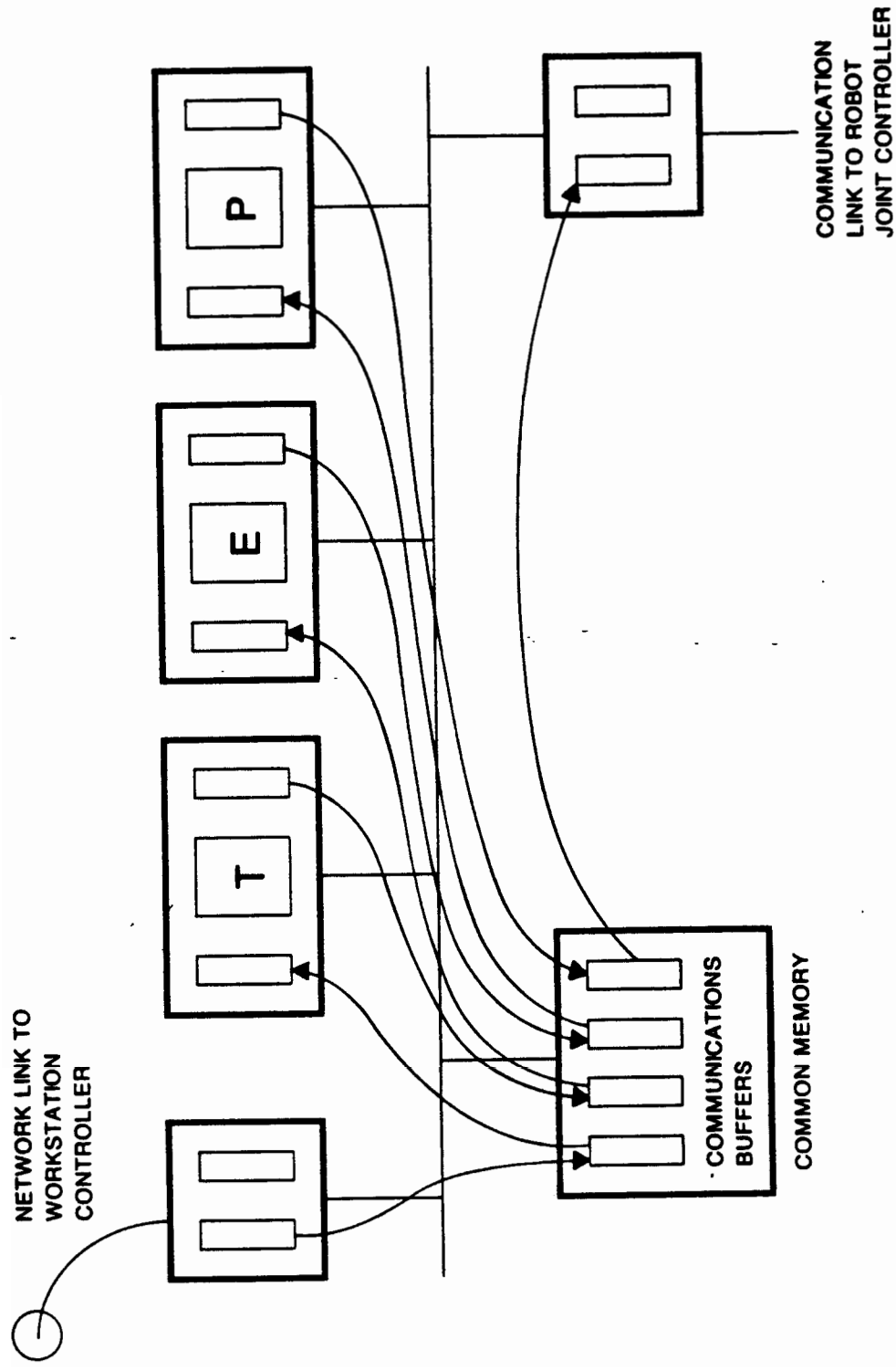


**The Input-Process-Output Structure  
is the Fundamental Building Block  
of the Architecture.**

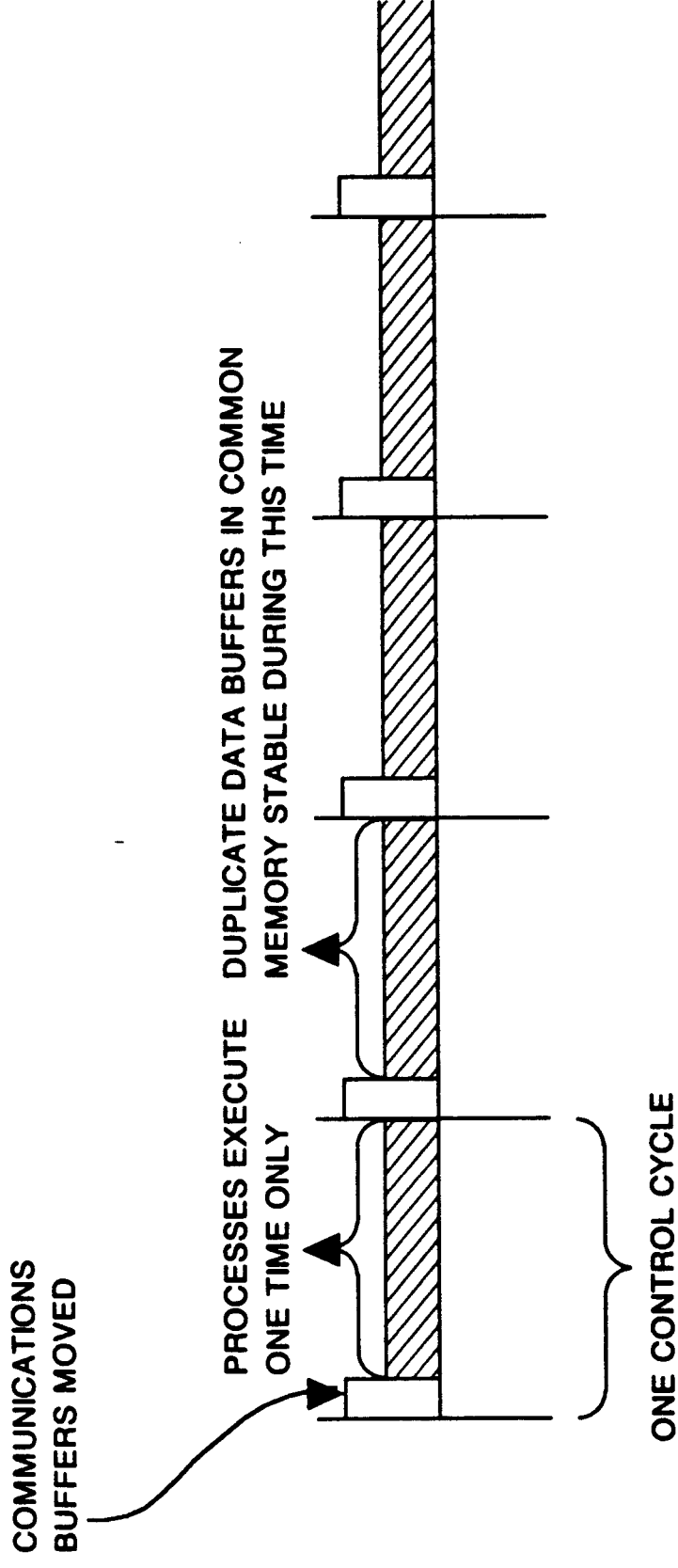
### **III.1**



A Standard Processing Format of Preprocess, Decision Process and Postprocess Is Used for Each Functional Module

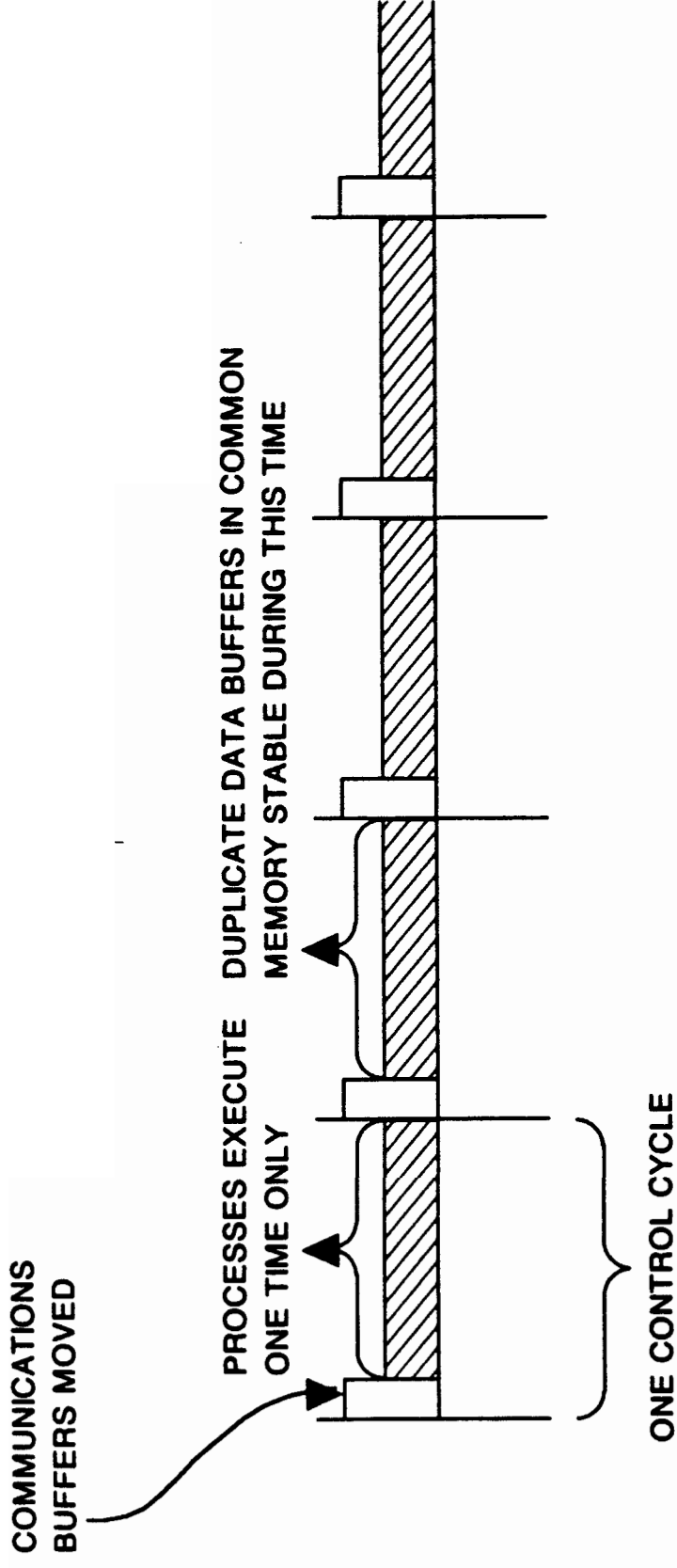


Communication of data between modules through a Common Memory eliminates process to process interactions and provides a snapshot of system status in the duplicate buffers in common memory for diagnostic evaluation.



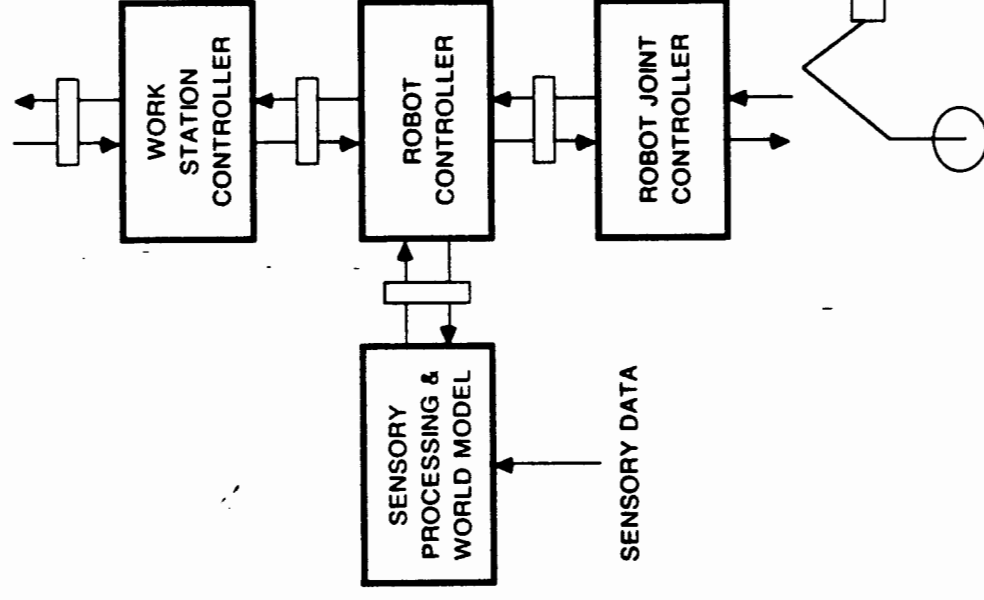
**Real-time behavior requires the continuous repetition of a control cycle that samples inputs and generates outputs.**

### III.4



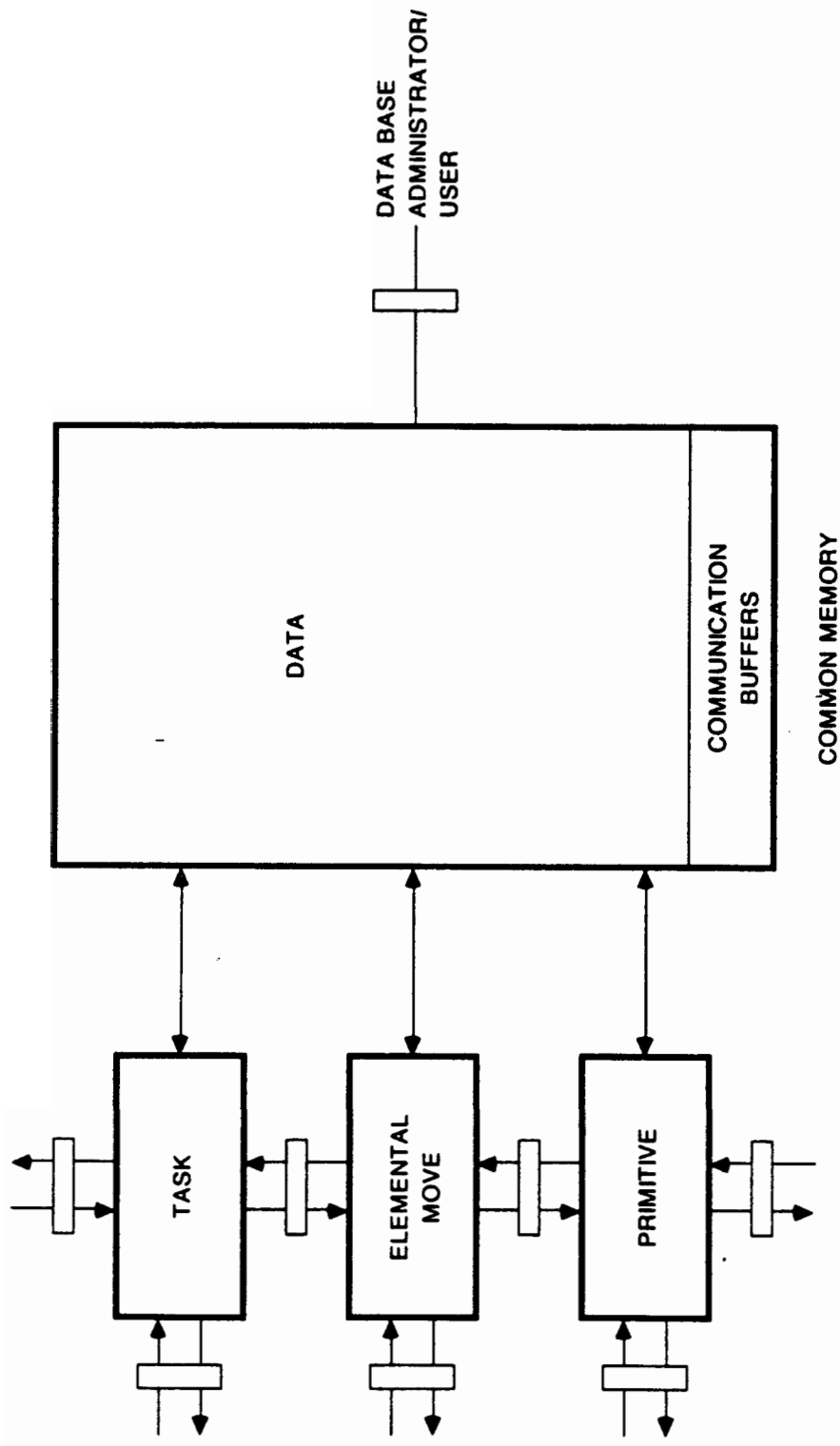
**Real-time behavior requires the continuous repetition of a control cycle that samples inputs and generates outputs.**

### III.4



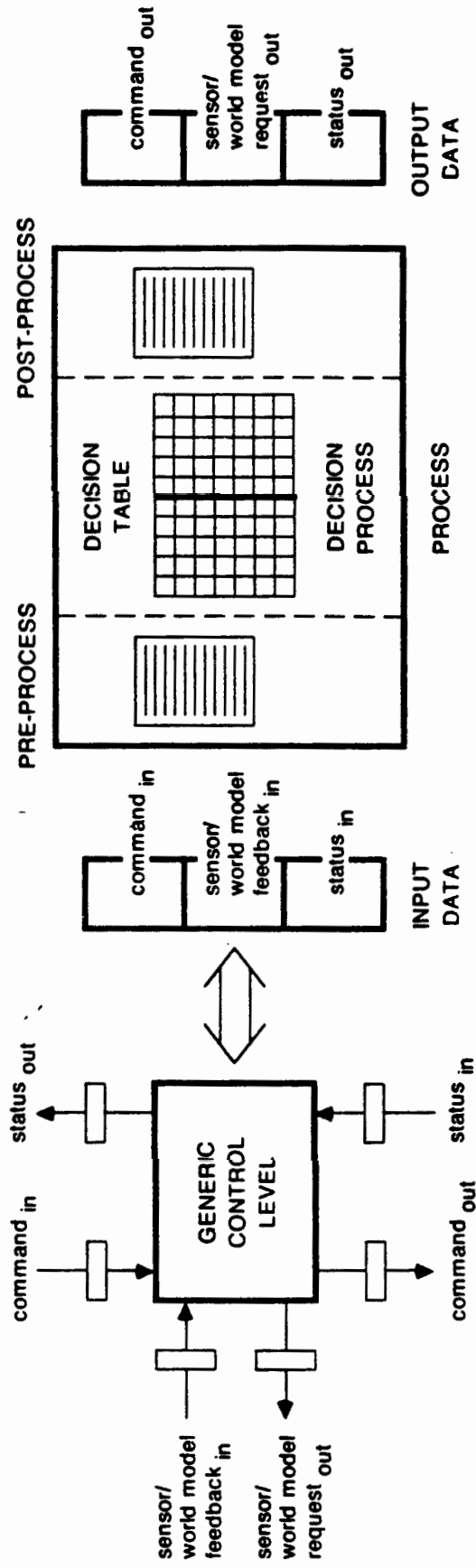
Robot Controller is integrated with other components in an Automated System through well-defined data interfaces.

### III.5



Robot Controller is composed of a set of generic levels that do hierarchical task decomposition. Task specific data is kept separate from the programs and is accessed in real-time from a knowledge base maintained in Common Memory.

### III.6



A Generic Control Level with interfaces to higher and lower level controllers and a World Model/Sensory Processing System is a generic processing structure.